基于区块链的可信云服务等级协议执行证人模型复现报告

张亦蕾

摘要

传统的云服务等级协议 (SLA) 缺乏可靠的自动执行平台。新兴的区块链技术为跟踪商业合作伙伴之间的交易带来了一个解决方案。然而，在将可能的违规行为记录到区块链上之前，证明 SLA 中可能存在的违规行为仍然非常具有挑战性。为了应对这一挑战，原论文提出了一个基于博弈论和智能契约技术的证人模型。本文利用以太区块链的智能合约复现了系统原型并验证其可行性和性能。复现实验的结果与原文的实验结果基本一致。

关键词：智能合约；区块链；云服务等级协议

1 引言

云计算是当今最流行的多租户共享资源的商业模式。云客户可以通过互联网灵活地使用远程提供商提供的计算、存储和各种其他资源作为服务。云 SLA 是客户和提供商之间就某些服务的质量达成的协议；在质量违规的情况下，客户将从提供商那里获得相应的补偿。在云计算的背景下，它是云客户和提供商之间关于云服务质量的协议。

Etherum 中的智能合同提供了一种通过区块链实现服务交易自动化和执行 SLA 的可行方法。然而，拟议的解决方案缺乏对已识别违规行为的可信度分析，并且在就区块链之外发生的事件达成共识方面仍面临挑战。已有的解决方案通常基于可信第三方，是集中式的，存在单点故障，很容易受到损害。

原文[1]引入了一个“证人”角色来改进现有的基于区块链的 SLA 解决方案，它以一种分散的方式检测具有明确可信度问题的违规行为。证人为系统中的匿名参与者，通过提供违规举报服务来获得收入。在原文的协议模型中设计了证人不同行为的支付函数，使得证人必须始终诚实地行事，以便为自己获得最大利润，并利用纳什均衡原理用博弈论证明了模型中证人的信任问题。在原文的证人模型中还提出了一种无偏随机排序算法，选择一定数量的证人 (通过服务提供商和客户之间的协商预先定义) 来组成委员会。委员会成员是随机挑选的，随机性不能由任何参与者主导。这对于避免大多数代表代表同一方 (客户或提供商) 的情况非常重要，以确保公平。此外，该算法还能够消除串通的机会，因为委员会成员不是预先确定的，他们之间没有事先认识的机会。

原文使用以太区块链的智能合同实施了一个原型系统，可以自动化实施 SLA 生命周期并增强角色之间的公平性，并在 Rinkeby 上进行了测试，证明了该证人模型的可行性和系统性能。其中，Rinkeby 是一个全球区块链测试网络，供开发人员调试开发的智能合约。本文对原文中的系统模型进行了复现，并在个人私链上部署了智能合约进行测试，实验结果与原文基本一致。

论文的其余部分组织如下。第二节讨论了云 SLA 和区块链的相关工作；第三节介绍了原文中的证人模型设计和关键技术；第四节详细介绍了复现实验的相关细节；第五节分析了原文实验和复现实验的结果对比；第六节对本次复现工作进行了总结，并对下一步工作进行了展望。

2 相关工作

SLA 是一个广泛讨论的研究主题，尤其是在云计算的背景下。它在服务提供商和客户之间建立了服务质量协议，在违反协议的情况下确保客户的利益。典型的 SLA 生命周期包括多个实施阶段，包括协商、建立、监控、违规报告和终止[2]。大多数研究工作集中在三个方面：(1) 二语习得术语和参数的语法定义，其目的是标准化表示，以便计算机系统可以在线处理二语习得；(2) 确保二语习得的资源分配技术。这方面的工作主要集中在优化资源分配的算法上，从而避免违反 SLA 的情况发生。此类工作中的 SLA 通常被视为约束；(3) 在 SLA 生命周期的特定阶段解决问题的系统或方法。但是其中最具挑战性的阶段————违规管理和报告，其相关研究较少。在行业中，Amazon CloudWatch 服务是提供商自动监控和通知的一个例子。在这种情况下，客户别无选择，只能信任供应商。Muller[3]开发了一个名为 SALMonADA 的平台来处理运行时的 SLA 冲突。它作为第三个可信任方执行监控和违规报告。所有这些工作都假设违规举报是可信的，但这是现实中最困难的部分。

智能合同被提出通过计算机协议来数字地促进、验证和执行合同[4]。一些探索，如[5]，将这一概念与云 SLA 协商结合起来，专注于智能合同的语义表达，以实现协商阶段的自动化。然而，他们中的大多数缺乏一个可信的平台来执行智能合同。这实际上很重要，因为智能合同依赖于一个强有力的假设，即没有人可以篡改它的执行。Town Crier[6]和 TLS-N[7]分别从硬件和传输协议级别确保了可信的执行和通信环境。但是，它们要么是集中式的，要么需要特殊的基础设施支持。

区块链上的交互是不可变的，因此它可以确保智能合约所要求的可信性。首先，Etherum[8]，即以太坊，实现了在其区块链上执行通用程序。Hiroki 等人[9]利用 Etherum 并设计了一组 Web API 来自动化区块链上的 SLA 生命周期实施。在他们的系统中引入了一个名为“服务性能监视器”的新角色，该角色负责违规报告。不过，对于发往区块链的违规报告是否可信，文中并未进行讨论。实际上，这对于基于区块链的系统来说仍然是一个空白：当事件发生在区块链之外时，如何可信地将随机事件记录到链上。目前，占主导地位的解决方案之一是使用“Oracle”来填补这一空白，一个代理充当区块链的“数据载体”。Oraclize 是一家受信任的公司，充当第三方，作为 Oracle 提供服务。但它存在单点故障，背离了区块链的去中心化思想。为了解决这个问题，Chainlink[10]在分布式先知上工作。分布式 Oracle 的工作方式是，只有在 Oracle 之间达成协议时，才能将事件的结果数据传送到链上或触发事务。这种理念仍然面临着一些弊端，如没有激励个人履行职责，要求个人独立和值得信赖，不同 Oracle 之间的共识问题等。

3 本文方法

本节介绍了在区块链上使用智能合同执行 SLA 的总体系统架构，然后详细说明了原论文的关键技术：无偏随机算法和智能合约的接口设计。

3.1 证人模型概述

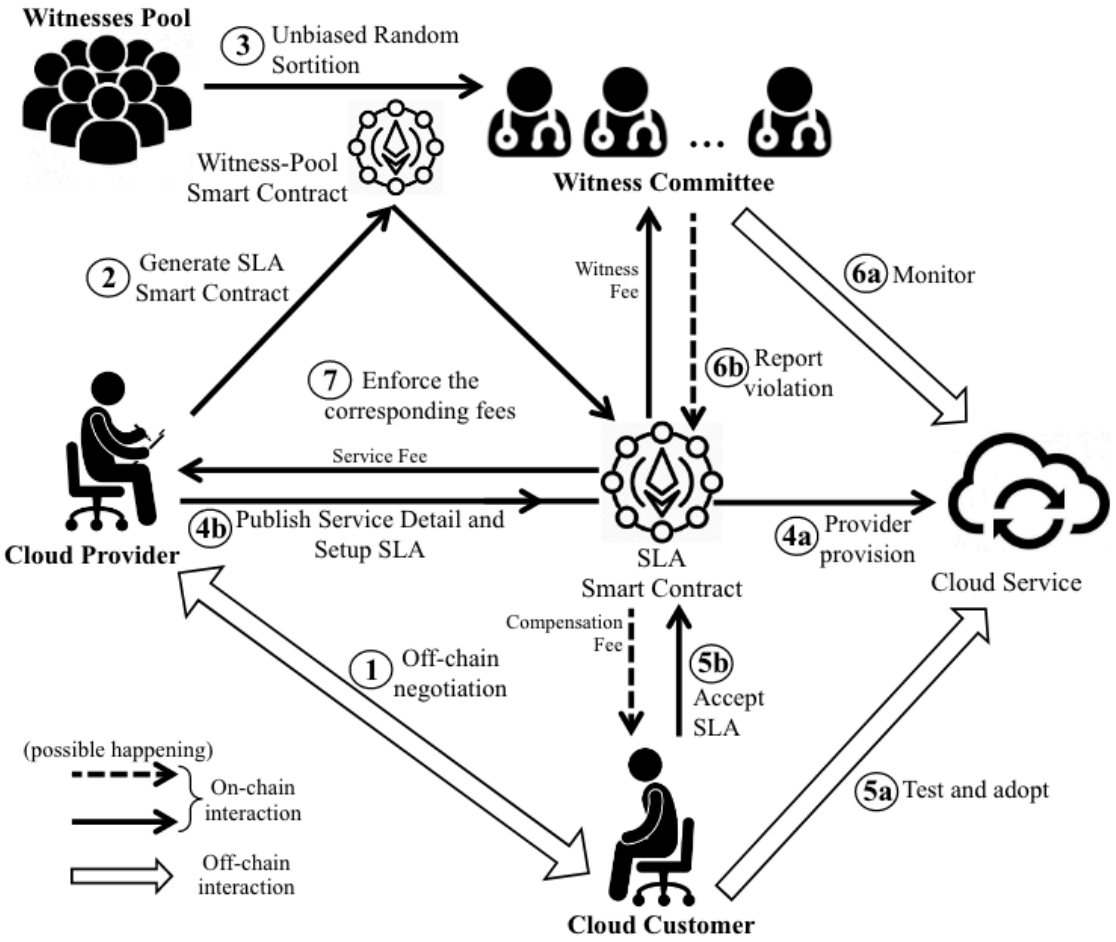

图 1: 云 SLA 实施的系统概述

图 1展示了原文为实施云 SLA 而设计的整体架构。该系统由两类基于区块链的智能契约组成：证人池的智能契约，这是系统的基础智能契约；特定 SLA 的智能契约，用于 SLA 的执行。对于证人池智能合同，主要有三个职责，包括证人管理、具体的 SLA 合同生成和证人委员会分类。任何拥有钱包地址的区块链用户都可以在证人池中注册其钱包地址，成为证人的成员。他们可以保持在线，并等待被选中参加某些特定的 SLA 合同。证人参与这一制度的动机是为了获得收益。证人参与系统越多，系统就越可靠、越可信。

整个 SLA 生命周期如下所示。提供商 P 首先利用证明池的智能约定为其自身生成 SLA 智能约定。在建立 SLA 之前，客户 C 应与提供商 P 协商详细的 SLA 条款，包括服务时间、服务费用、补偿费用等。其中，最重要的条款之一是确定 N，即为执行该 SLA 而雇用的证人的数量。SLA 中涉及的证人越多，违规检测结果就越可信。然而，另一方面，支付的证人费用越多，客户和提供商都需要平等地支付这笔费用。根据协商结果，提供商能够定制这些参数并生成新的 SLA 智能合同。之后，可以通过第 3.2 节详细介绍的排序算法，选择一组 N 个证人成员组成证人委员会，该算法也是通过区块链上的证人池智能合约来实现的。该算法被设计为无偏见和随机的，保证在委员会中挑选的证人是独立的，不会属于特定的一方，无论是 C 还是 P，实现了双方的互相信任。同时，提供商提供其云服务以供客户使用，并能够在 SLA 智能合同中发布其服务细节。证人委员将根据这些细节开始监测服务。自第一次违规报告以来，智能合同将开始计算一个时间窗口，T_{report}。在此时间窗口内，智能合同接受来自其他证人的报告。当时间窗口 T_{report} 结束时，如果智能合同从证人委员会收到的 N 个报告中有不少

于 M 个报告，则自动确认违规。M 也是由 P 和 C 协商的。然后在 SLA 智能合同中定义它。当然，M 必须大于 N 的一半。此外，M 越大，违规确认越可信。例如，如果委员会中有 N=3 名证人，则只有当至少 M=2 名证人报告该事件时，才能确认违反服务。在某种意义上，这 N 个独立证人构成了一个 n 人博弈，每个证人都想最大化自己的收益。原文中特别设计了保证证人诚实行事的支付函数，并利用博弈论的纳什均衡原理证明了证人在这场博弈中必须是一个诚实的玩家。也就是说，他们必须根据真实事件来报告违规行为。最后，SLA 在两种情况中结束：一是服务时间 Tservice 结束，没有违规；另一种情况是违反了 SLA。根据不同的情况，这三个角色可以从 SLA 智能合同中提取相应的费用。

3.2 无偏随机排序算法

在证人模型中，对特定 SLA 合同的证人选择必须是公正的，即提供商和客户都不能在委员会的选择中占据优势，这一点至关重要。模型中已经引入了以太作为可信方，原文提出了一种用于委员会选择的无偏随机排序算法，如算法 1 所示，该算法在证人池智能合约中实现。

Procedure 1 Unbiased Random Sortition.

Input:
Registered witness set, *RW*, a list of addresses;
The the size of the list, *len(RW)*;
The number of online witnesses, *oc*;
Required number, *N*, of members in a witness committee;
The hash value, B_b^{hash}, of the b^{th} block B_b at request;
The block index, *Id*, of current block;
Following sequential, K_s, blocks;
Confirmation, K_c, blocks;
The address of the provider, *p.address*;
The address of the customer, *c.address*
Output: Selected witness set, *SW*, to form a committee.

assert($Id > b + K_s + K_c$) && assert($oc >= 10 * N$)
 seed $\leftarrow 0$
 for *i* ***in*** K_s **do**
 | seed + = B_{b+1+i}^{hash}
end
while $j < N$ **do**
 | $index \leftarrow$ seed % $len(RW)$
 | **if** *RW [index].state == Online*
 | *&& RW [index].reputation > 0*
 | *&& RW [index].address != c.address*
 | *&& RW [index].address != p.address* **then**
 | | $RW[index].state \leftarrow Candidate$
 | | oc $--$
 | | Add $RW[index] \Rightarrow SW$
 | | $j++$
 | **end**
 | $seed \leftarrow hash(seed)$
end
return *SW*

首先，证人池智能合同作为基础合同来管理证人。它为任何区块链用户提供了一组注册到池中的接口。登记后的用户能够将其状态转换为“Online”或“Offline”，以指示何时可以选择它。详细的证人状态管理见 3.3 节。证人池中的地址以注册顺序中的列表进行管理。智能契约中设计了两个接口，用于从池中选择 N 个见证人。在块 B_b 处，首先由某个特定的 SLA 智能合同调用“Request”接口。这

意味着该事务将被记录于第 b 个块的索引中，该块的散列值是 B_b^{hash}。在区块链生成 K 个块后，证人池智能合同将调用另一个接口“Sortition”来选择 N 个在线用户作为候选证人。排序算法如上所示。

它将上述 K 个块中前 K_s 个的哈希值作为 seed。此外，我们还需要等待其他 K_c 块来确认所采用的块，其中 $K_s + K_c = K$。

- K_s 用于确保某一方顺序产生 K_s 块的概率很小。
- K_c 用于使之前的 K_s 候选块最终以主导概率参与主链。
- 这两个值取决于区块链自身的属性。考虑到以太主网,[11]显示，排名前四的矿商控制着 61% 的采矿权。因此，原文中建议 $K_s = 10$，这样即使前四名矿工串通，seed 也有 99% 以上的可能性无法操纵和预测，而 K_c 被普遍认为应该取值为 12。

seed 只从证人地址池列表中选择声誉积极的在线用户并基于先前 seed 的散列值生成新的 seed。重复这一过程，直到选定所需的 N 名证人。在算法 1 的开始，首先检查在线用户数，*oc*，至少有所需的证人数，N，的 10 倍，这保证了证人选择的随机性。

考虑到产生一个块的哈希值和组合连续的 K 个块作为 seed 本身的困难，我们可以证明排序算法是随机的和无偏的，即供应商和客户都不能操纵选择结果来在委员会中占有优势。

3.3 原型接口

根据证人模型和支付函数设计，原文实现了一个基于以太智能合约的原型系统。在 SLA 执行系统中有三个角色和两种类型的智能合同。角色包括传统的供应商和客户，以及引入的证人。智能合同包括证人池智能合同和 SLA 智能合同。在两个智能合约中设计的界面都被命名为图 2和图 3中箭头上的文本。文本的格式为“$R_{role} \rightarrow [C_{type} ::] N_{interface}$”。这意味着只有角色 *R* 可以调用接口 *N*，该接口在智能合约 *C* 中定义。角色的表示为 *W* 表示证人，*P* 表示提供商，*C* 表示客户，*SC* 表示生成的 SLA 智能合同。为了表示 C_{type}，*WP* 用于见证池类型的智能契约，而 *SLA* 用于生成的智能契约以实施特定的 SLA。

图 2说明了智能契约中定义的见证角色的状态。包括“Online”、“Offline”、“Candidate”和“Busy”四种状态。只有注册后的用户才能将其状态转换为“Online”，之后它可能会被 SLA 智能合同选中，因此用户需要在区块链上持续监控自己的状态。一旦它被证人池智能合约通过执行排序算法 1 选中，它的状态就变成了“Candidate”，即证人候选人。在确认时间窗口 (例如，2 分钟) 内，证人候选人可以查看 SLA 智能合同，并决定是确认还是拒绝该任命，如果拒绝，SLA 智能合同的提供商必须再次请求执行证人选择算法；若接受，它的状态将变成“Busy”。在每个 SLA 生命周期迭代结束时，证人有权通过接口“witessRelease”主动离开 SLA 合同。除此之外，提供者可以调用接口“setSLA”来解散证人委员会，用户将被动地解除证人身份。

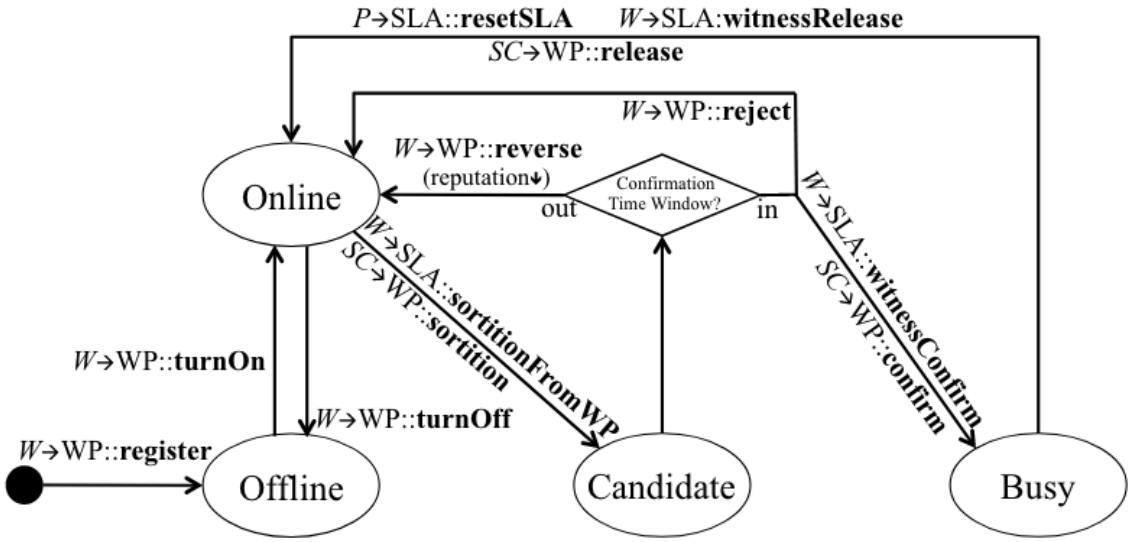

图 2: 证人池智能合同中用户的状态转换图

为了防止一些恶意行为，原文引入了信誉值来衡量每个证人的行为。首先，每个证人在注册时都有一个初始信誉值 R_{init}，它可以是一个预定的常量，也可以是一个取决于其所持股份或某些存款的值。例如，当某些用户实际上不可用或不频繁检查其状态时，该用户的状态仍为“Online”，然而，当该用户被某个 SLA 智能合同选择时，它将无法在确认时间窗口内确认选择和加入。在这种情况下，因为它的状态为“Candidate”，要被动地转为“Online”状态需要调用接口“Reverse”，其信誉值将下降 10%。如果该值变为零或更小，则它将被永久拉黑。

图 3显示了特定 SLA 智能合同实施的生命周期状态转换。SLA 智能合同由提供商发起生成，共有五种状态：“Fresh”、“Init”、“Active”、“Violated”和“Completed”。虚线箭头显示了服务违规时的状态转换路径。图中的三个方块表示此智能合同中的相应角色，在 SLA 结束时，合约将自动向各用户执行转账。

其中，区块链上的智能合约中的状态转换必须由接口触发，执行起来需要一些成本。因此，在某些情况下，角色是执行接口的最大的受益者，即有动力进行状态转换。例如，当服务时长正常结束时，提供者是获得全部服务费的最大受益者。它必须积极利用接口“ProviderEndNSLAandWithDrawing”来结束正常的 SLA 并收回自己的收入。同时，将预付金作为支付函数设计划分给不同的证人。之后，其他角色可以提取自己部分的收入。类似地，当发生违规行为时，客户是最有动力获得补偿费的人。它可以利用“CustomerEndVSLAandWithDrawing”来结束违反的 SLA，并将状态从“Violated”转换到“Completed”。

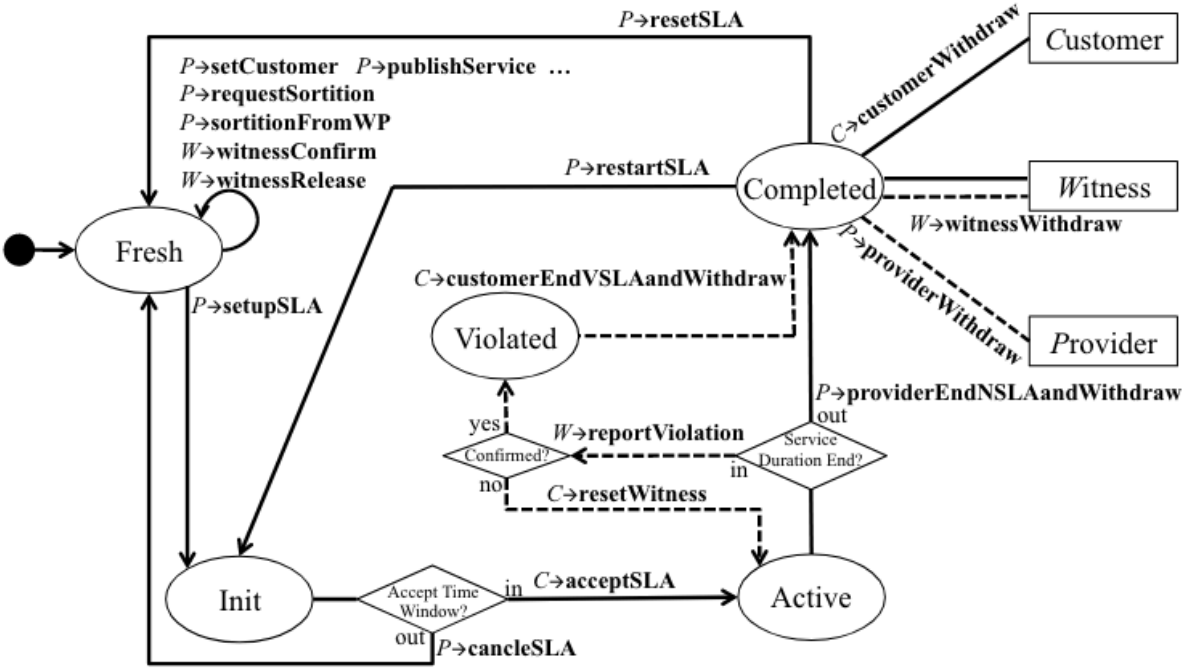

图 3: 特定 SLA 智能合同的 SLA 生命周期状态转换图

4 复现细节

4.1 与已有开源代码对比

由于原论文[1]没有公布源代码，本文根据原文给出的接口功能和状态转换图进行了证人系统的复现工作。

本文利用 Etherum 提供的编程语言 Solidity 对原文提到的所有接口按照其功能逻辑进行编程，能够实现 3.3 中给出的所有用户状态转换和 SLA 生命周期完整运行，包括 SLA 生成、自动转账和证人选择等重要功能。算法 1，即证人选择功能的实现代码如图 5所示。

```
function sortition(uint B_b, uint N, address customer, address provider)
external
    //returns (address[] memory SW)
{
    require(block.number > (B_b + K_s + K_c));
    require(oc >= (10 * N));

    emit Sorted();

    uint seed = 0;
    for(uint i = 0; i < K_s ; i++){
        bytes32 _hash = blockhash(B_b + i + 1);
        uint _uint_hash = uint(_hash);
        seed += _uint_hash;
    }

    uint j = 0;
    uint index;
    uint len = WP.length;
    while(j < N){
        index = seed % len;
        address _address = WP[index];
        User storage _index = users[_address];
        emit SLAStateCheck(_index.state);
        if ( _index.state == State.Online
        && _index.reputation > 0
        && _address != customer
            && _address != provider
        ){
            emit OneWitnessSorted();
            _index.state = State.Candidate;
            oc--;
            SW[j] = _address;
            j++;
            //发送证人确认窗口，开始窗口计时
        }
        bytes32 _bytes32Hashseed = keccak256(abi.encodePacked(seed));
        seed = uint(_bytes32Hashseed);
    }//candidates sorted
```

图 4: 算法 1 的实现代码

4.2 实验环境搭建

智能合约的部分采用 Remix IDE 进行编译和部署，运行环境绑定 Ganache 创建的私链工作面板，工作面板的用户列表如图 5所示。

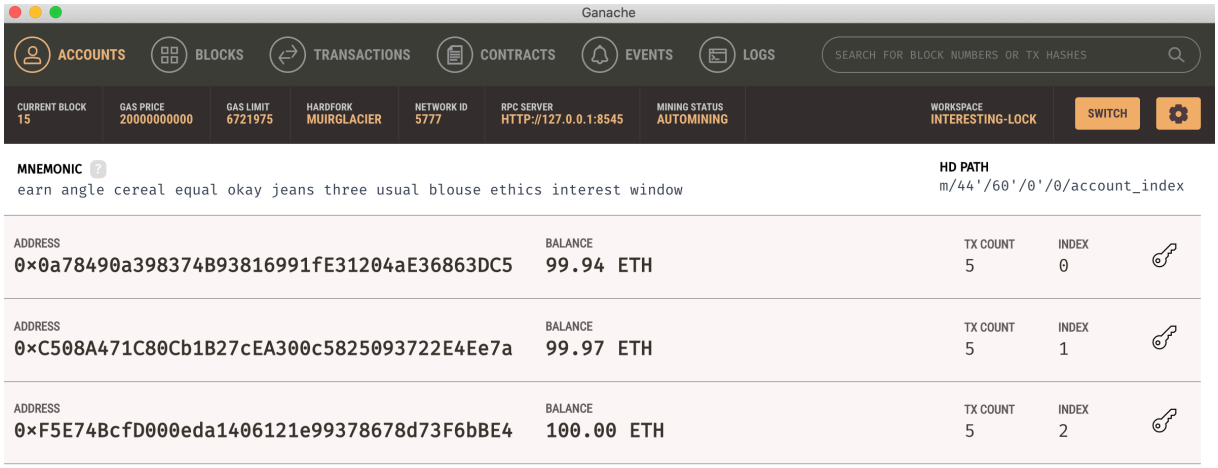

图 5: Ganache 工作面版用户界面

4.3 界面分析与使用说明

图 6为 Remix 的运行界面，用于绑定环境和部署合约。图 7展示了已部署的证人池智能合约和 SLA 智能合约以及 SLA 智能合约中的部分接口。本文通过在图 6中的用户窗口选择用户身份再点击图 7中的函数接口来执行一次事务，记录其 Gas 消耗量来作为接口性能测试的结果。因为矿工需要执行接口中定义的程序进行验证，接口越复杂，调用时所需的交易费用就越高。这在以太中定义为“Gas”，是一个单位，指的是矿工在执行交易时所做的工作量。

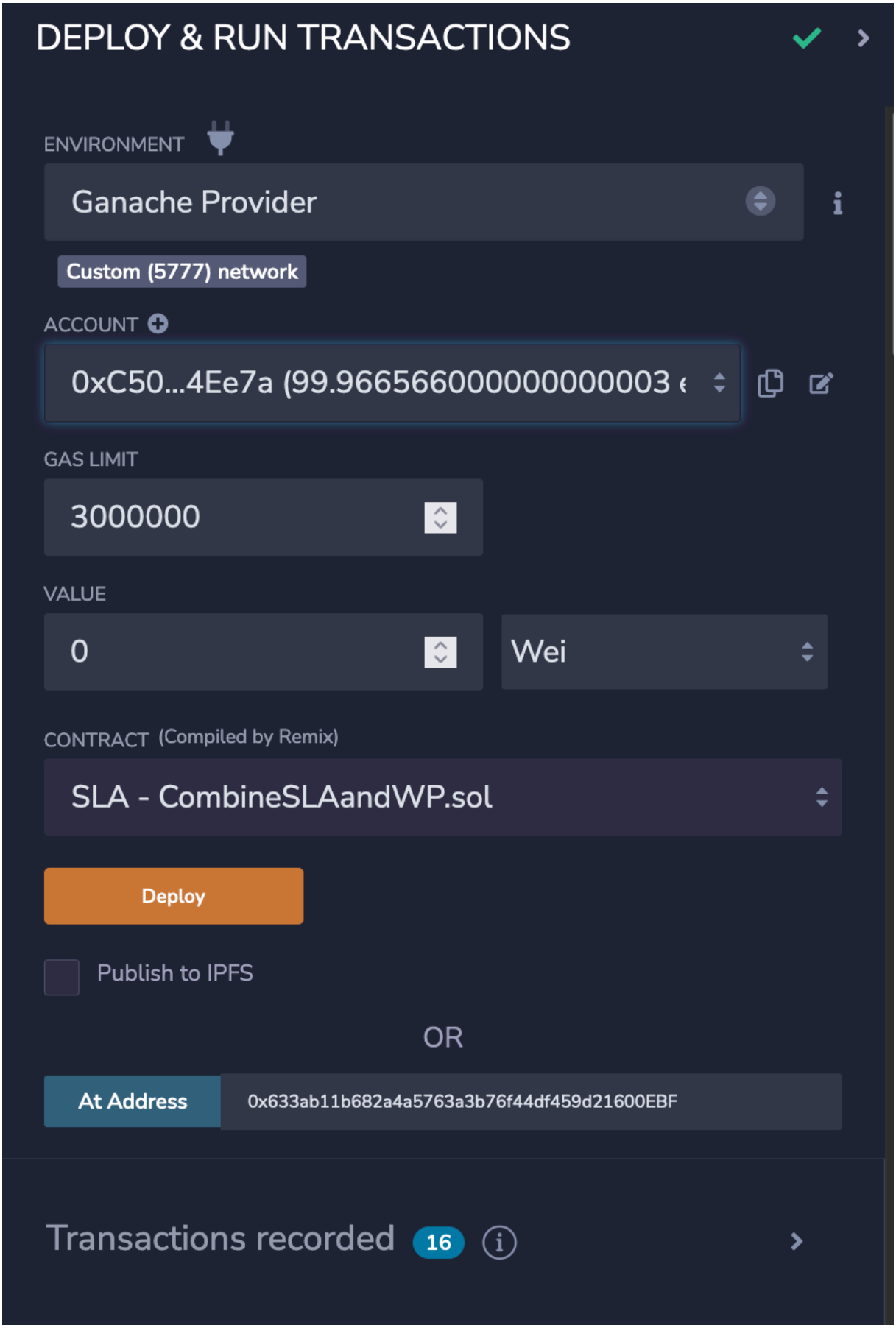

图 6: Remix IDE 的运行界面

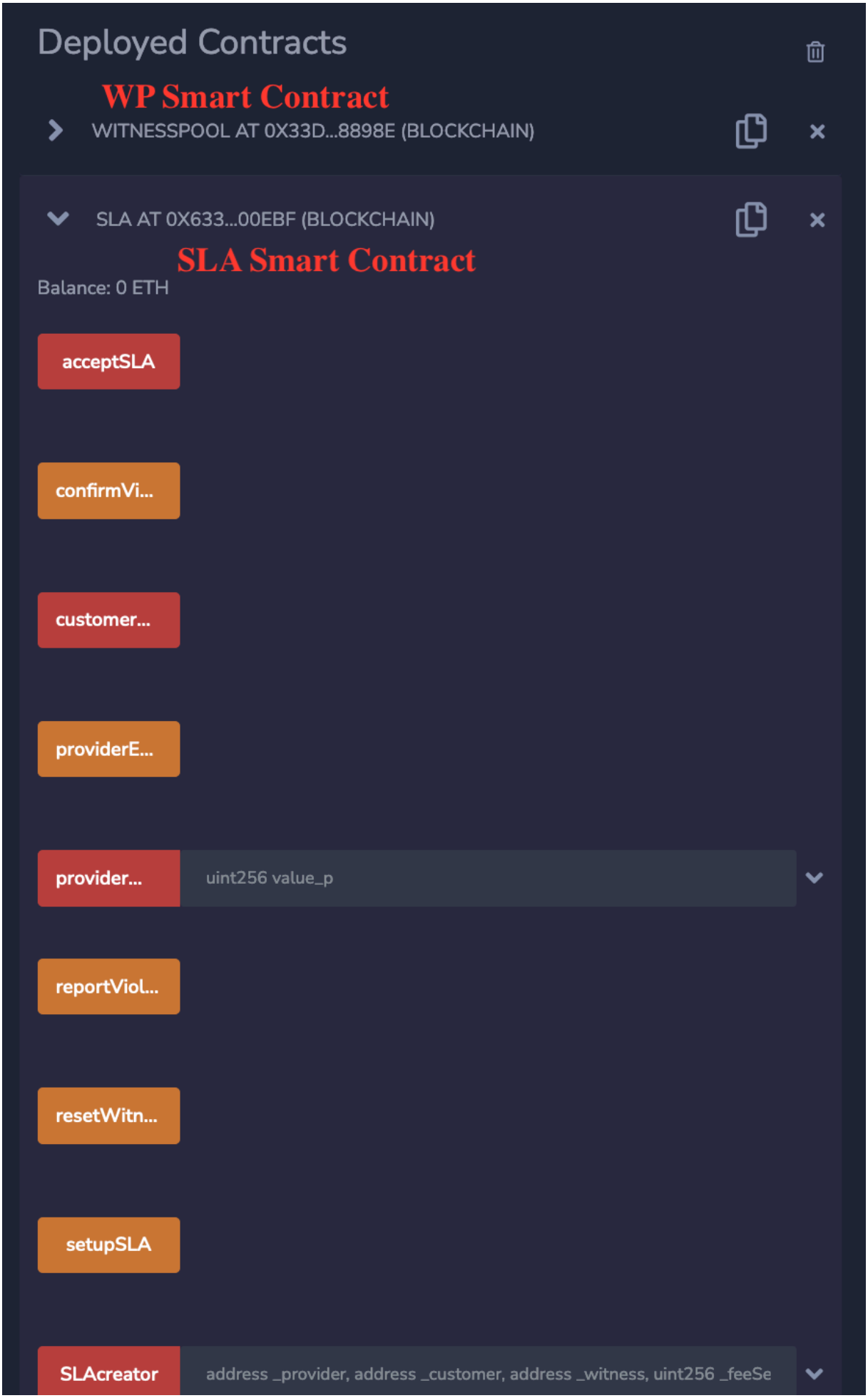

图 7: 已部署的两个合约和部分接口展示

5 实验结果分析

原文将实现的智能合约部署在以太区块链的测试网–“Rinkeby”上。它是一张覆盖全球的区块链测试网，供开发者调试智能合约。作为以太的加密货币，以太在测试网上没有实际价值，可以申请进行调试。系统的信任问题部分由博弈论证明，并由无偏排序算法保证，其可信度由区块链技术背书。因此，原文主要对实验研究中的一些性能信息进行分析。无论是在试验网还是在主网上，耗气量都是相似的。

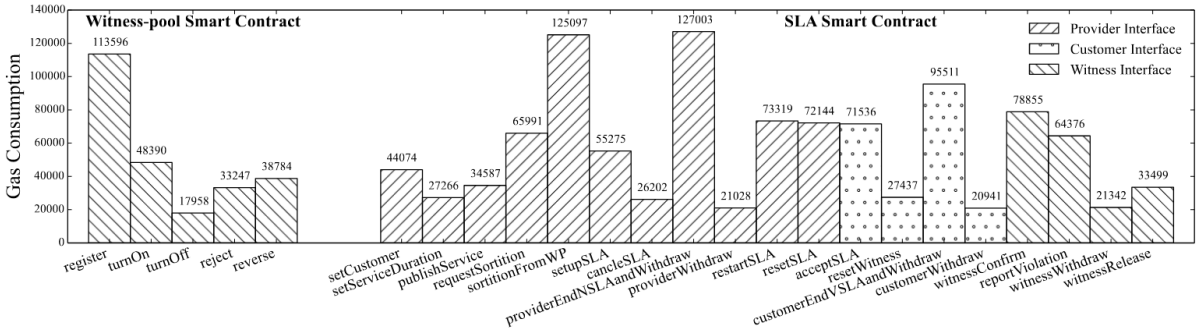

图 8: 原文实验结果：智能合约中各界面的耗气量

图 8为原文实验研究的结果。可以看到，与客户和见证人相比，提供商在整个 SLA 生命周期中需要更多的“Gas”。客户和见证人的接口消耗较少。这符合初始的模型设计和实际情况。因为在大多数情况下，提供商通过提供服务赚取最多的收入。它有继续进行生命周期的动机。证人角色的“Gas”消耗量为轻量级的，更能够说服区块链用户参与到系统中成为证人。

本文在 Ganache 上生成了一条私链的工作面板，并创建三个帐户来模拟不同的角色，即提供商、客户和证人，如图 5所示。主要利用在每个模拟帐户上的‘Ether’来执行接口，并根据模型预付不同类型的费用。为了进行实验，本文首先部署了证人池智能契约作为基础，并将所有帐户注册到证人池。然后，提供商生成 SLA 智能合同，开始与客户的 SLA 生命周期。本文复现了原文的实验，测试所有可能的场景，验证不同接口的功能，从实验的交易历史中记录每个界面的所有气体消耗。实验结果如图 9所示。

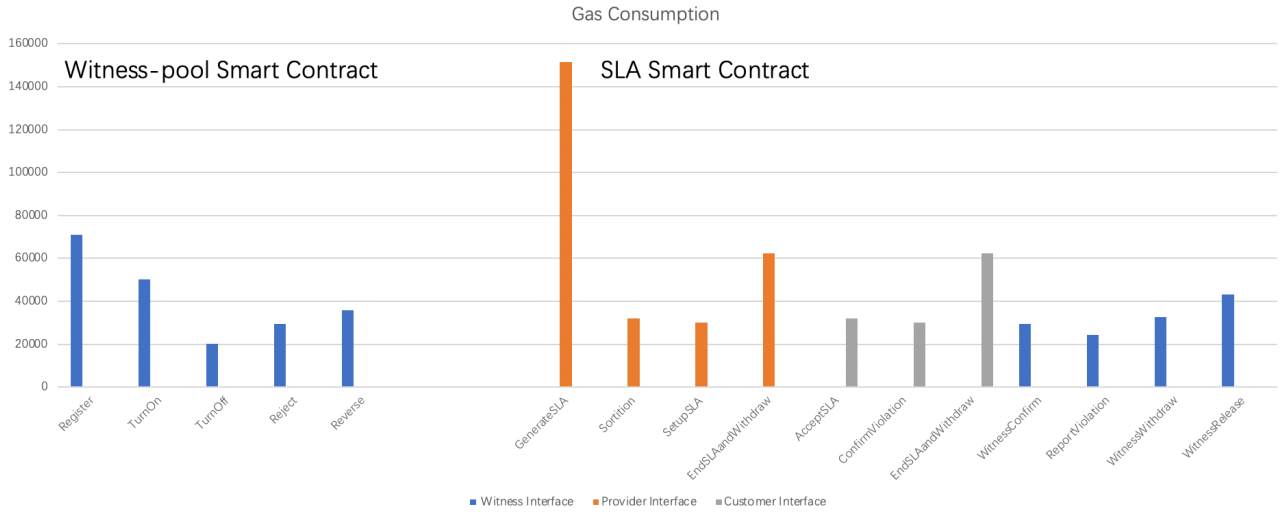

图 9: 复现实验结果

可以看到大部分接口的“Gas”消耗量都与原文近似。服务商创建 SLA 合同这一接口的“Gas”消耗量最大，且在整个 SLA 生命周期中，属于服务商的接口“Gas”消耗最多。这与原文的实验研究结

果是相同的。图 10展示了部分交易记录，即接口性能实验数据。

图 10: 部分交易事务记录

6 总结与展望

本文介绍了文章[1]中提出的协助云 SLA 协议实施的证人模型。在没有源码可以参考的情况下，本文利用以太智能合约对该证人模型进行复现工作，实现了一个原型系统。不仅实现了 SLA 实施生命周期，还实现了证人池的证人管理。实验部分复现了原文关于系统性能的研究，结果与原文基本一致。对于未来的工作，仍有可能进一步优化接口实现，以降低“Gas”消耗耗量，丰富智能合约的功能。此外，可以考虑更多的场景来应用该证人模型。

参考文献

[1] ZHOU H, OUYANG X, REN Z, et al. A blockchain based witness model for trustworthy cloud service level agreement enforcement[C]//IEEE INFOCOM 2019-IEEE conference on computer Communications. 2019: 1567-1575.

[2] FANIYI F, BAHSOON R. A systematic review of service level management in the cloud[J]. ACM Computing Surveys (CSUR), 2015, 48(3): 1-27.

[3] MÜLLER C, ORIOL M, FRANCH X, et al. Comprehensive explanation of SLA violations at runtime [J]. IEEE Transactions on Services Computing, 2013, 7(2): 168-183.

[4] CLACK C D, BAKSHI V A, BRAINE L. Smart contract templates: foundations, design landscape and research directions[J]. arXiv preprint arXiv:1608.00771, 2016.

[5] SCOCA V, URIARTE R B, DE NICOLA R. Smart contract negotiation in cloud computing[C]//2017 IEEE 10Th international conference on cloud computing (CLOUD). 2017: 592-599.

[6] ZHANG F, CECCHETTI E, CROMAN K, et al. Town crier: An authenticated data feed for smart

contracts[C]//Proceedings of the 2016 aCM sIGSAC conference on computer and communications security. 2016: 270-282.

[7] RITZDORF H, WÜST K, GERVAIS A, et al. TLS-N: Non-repudiation over TLS enabling-ubiquitous content signing for disintermediation[J]. Cryptology ePrint Archive, 2017.

[8] BUTERIN V, et al. A next-generation smart contract and decentralized application platform[J]. white paper, 2014, 3(37): 2-1.

[9] NAKASHIMA H, AOYAMA M. An automation method of sla contract of web apis and its platform based on blockchain concept[C]//2017 IEEE International Conference on Cognitive Computing (ICCC). 2017: 32-39.

[10] ELLIS S, JUELS A, NAZAROV S. Chainlink: A decentralized oracle network[J]. Retrieved March, 2017, 11: 2018.

[11] GENCER A E, BASU S, EYAL I, et al. Decentralization in bitcoin and ethereum networks[C]// International Conference on Financial Cryptography and Data Security. 2018: 439-457.